An Agent-Based Financial Market Simulator for Evaluation of Algorithmic Trading Strategies

Rui Hu
Quantica Trading
119 King St. West Suite 300
Kitchener, Ontario, Canada
Rui@quanticatrading.com

Stephen M. Watt
Computer Science Department
University of Western Ontario
London, Ontario, Canada
Stephen.Watt@uwo.ca

Abstract—**Algorithmic trading strategies are most often evaluated by running against historical data and observing the results. This limits the evaluation scenarios to situations similar to those for which historical data is available. In order to evaluate high frequency trading systems in a broader setting, a different approach is required. This paper presents an agent-based financial market simulator that allows the exploration of market behaviour under a wide range of conditions. Agents may simulate human and algorithmic traders operating with different objectives, strategies and reaction times and market behaviour can use combinations of simulated and historical data. The simulator models the market's structure, allowing behaviours to be specified for market makers and liquidity providers and other market participants. The primary use of the system has been in the evaluation of algorithmic trading strategies in a corporate setting, but other uses include education and training as well as policy evaluation.**

Keywords*–*Agent-Based Simulation, Financial Markets, High Frequency Trading

I. Introduction

Algorithmic trading has grown rapidly around the globe and has dramatically changed how securities are traded in financial markets. According to a few reports [1]–[3], more than 50% of the volume of U.S. equity markets in recent years has been generated by algorithmic trading. In order to manage risk exposure and optimize profit, algorithmic trading strategies are generally evaluated for correctness and performance before launching them in real markets. This is carried out in practice through simulators. Existing simulators significantly rely on real-time market data or historical data, typically recorded from actual market for the purpose of back-testing trading models during their development cycle. While this can provide traders with valuable information, there are a number of pitfalls. First, live market data is not always available, which restricts the use of simulators to certain market hours. In addition, the trading strategies tested do not have any impact to the market as they can only follow the trend and their orders are simply executed based on the current market conditions. Similar issues also exist in back-testing approaches where trading strategies are tested against existing data set with the problematic assumption that the orders would not have changed the historical prices if they were executed in the real market. Moreover, this approach is limited by potential over-fitting. By refining the parameters of a trading strategy on a particular period of historical data, the results can become skewed and produce returns that can never work again. Last, but not least, existing simulators typically do not provide a standard protocol for interaction with users. Instead, they require skills in specific programming languages and demand trading strategies to be implemented on top of proprietary Application Program Interfaces (APIs). This can restrict the evaluation of trading strategies to a single simulation environment. This issue can be seen in TT Sim [4].

We are motivated by the question of how to design and implement a simulator that can support market simulation research and is suitable for evaluation of algorithmic trading strategies. The simulator should be independent of any particular data feed and be able to provide a realistic testing environment by reproducing certain phenomena of a real market. This would be beneficial to strategy testing as some phenomena, such as the flash crash [5], are hard to predict and do not occur often in real markets. Such phenomena can, however, be imitated easily in the simulation environment. Multiple users should be able to connect to the simulation server at the same time, allowing them to not only assess the viability of their trading strategies using pre-defined market conditions, but also to create very specific ones that suit their needs.

We present such a simulator that is intended to be useful for evaluating trading strategies. It supports a variety of security types, including equities, futures, foreign exchange, and options. The simulator may run a market consisting exclusively of simulated (human or robotic) trading agents using different trading strategies (e.g., to study algorithms or market effects), or external participants (typically human) may log in and interact with the simulated market (e.g., for training). Participants (logged in users or simulated trading agents) can submit both limit and market orders with different time-in-force, allowing them to interact with the simulation environment as if they were trading in a real market. At the same time, the simulator adopts the Financial Information eXchange (FIX) protocol [6], an open standard that is used extensively by global financial markets. This allows multiple users to interact with the simulation environment simultaneously and independently. In contrast to other simulators that require testing trading strategies to be built on top of proprietary APIs, our simulator uses open protocols so can be integrated easily into their systems with little modification. The simulator is able to run in two different settings, each of which is useful in certain scenarios.

The first setting uses simulated trading agents, each of which is able to adapt and react to real-time market events by following selected pre-defined strategies. All of the agents are configurable. By adjusting their configurations, we can create very specific market conditions that would occur only rarely in a real market. In addition, the agent-based simulator is able to run at any time as the data are generated by the computerized agents. The agent-based simulator is useful in that it allows

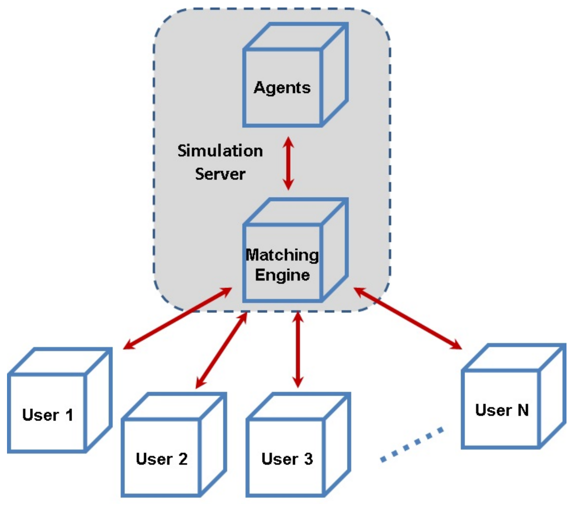
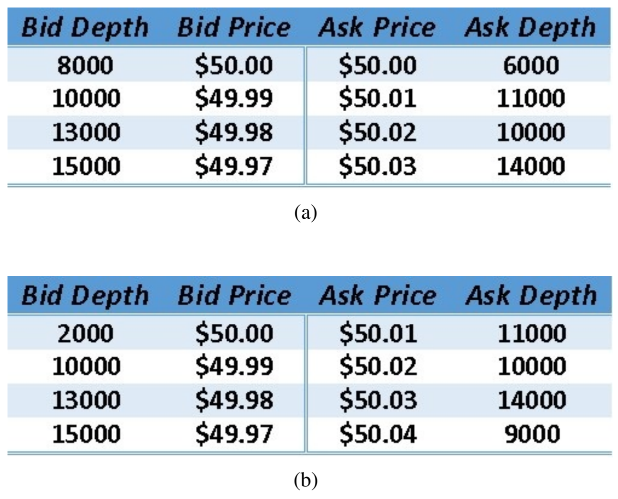


Figure 1: Agent-based simulator architecture.



Bid Depth	Bid Price	Ask Price	Ask Depth
8000	$50.00	$50.00	6000
10000	$49.99	$50.01	11000
13000	$49.98	$50.02	10000
15000	$49.97	$50.03	14000

(a)

Bid Depth	Bid Price	Ask Price	Ask Depth
2000	$50.00	$50.01	11000
10000	$49.99	$50.02	10000
13000	$49.98	$50.03	14000
15000	$49.97	$50.04	9000

(b)

Figure 2: Order book snapshots: (a) Before an order match (b) After an order match.

users to create desired market conditions where they can test their trading strategies whenever it is needed.

In the second setting, the simulator receives live market data from real exchanges and broadcasts this data to each user. Orders that are submitted by users are executed based on the current market conditions. This type of simulator is claimed by some to be more realistic. Meanwhile, the real market data simulator, in practice, has access to all the products available in the markets, while in agent mode this is not feasible unless there is a further configuration. We provide both these settings to users to evaluate their trading strategies, and give them the freedom to choose the one that is most suitable for their needs. A demo of our simulator is available for download at [7].

The remainder of this article is organized as follows: Section II presents the architecture of the simulator. Section III discusses the more important design and implementation details. Section IV illustrates a variety of scenarios where the simulator is found useful. Section V compares the current approach with related work. Finally, Section VI presents conclusions and discusses directions for future work.

II. SIMULATOR ARCHITECTURE

We now present the framework of the simulator which allows logged in traders to interact with artificial intelligent agents. The framework currently consists of a matching engine, a communication interface and a variety of simulated trading agents. Figure 1 shows a high-level overview of the architecture of the simulator. In the setting of using live market data, the agents will be replaced with data streams from real exchanges.

A. Matching Engine

The core of the simulator is a matching engine that accepts in-bound orders from both logged in users and computerized agents. It maintains a number of order books, each of which records the interest of buyers and sellers in a particular security and prioritizes their orders based on their price and arrival time (e.g., first come first serve). Buy orders with the highest price and sell orders with the lowest price are placed at the top. This centralized order system continuously attempts to pair buy and sell orders and trades are announced when certain matching rules are satisfied.

Matching rules are implemented based on continuous double auctions. An incoming buy order is first compared with the best sell order in the order book. If there is a price match, a trade will be generated at the best price. The matching engine will then send execution reports to the issuers of the two orders and announce a trade to all market participants. If the new order is not completely filled, the matching engine will try to fill the remainder of the order with the next highest ranking order available from the opposite side. This procedure is continued until either the new order is completely filled or there are no more matches. Figure 2 shows snapshots of an order book before and after an order match.

The matching engine also publishes quote updates to all subscribers. This is a major challenge as the matching engine must be able to process a high throughput of data with very low latency and broadcast updates to many subscribers at once. This requires us to design and implement a highly efficient financial information protocol for communications.

B. Communication Interface

The Financial Information eXchange (FIX) protocol is an electronic communication protocol, first introduced in 1992, whose primary objective is to exchange real-time stock trading data between entities. It has experienced a tremendous growth over the past years and has become the *de facto* communications standard in global financial markets. FIX messages are constructed with a number of fields, each given by a tag-value pair and separated by a delimiter. Figure 4 shows an example of a FIX message.

The simulator may run a market consisting exclusively of simulated trading agents running algorithmic strategies (e.g., to study algorithms or market effects), or participants (typically human) may log in and interact with the simulated market (e.g., for training). To allow multiple users to interact with our simulator simultaneously and independently, we use the FIX protocol. The simulator provides each user a designated port for login and maintains a dedicated channel for communication. Upon receiving a FIX message, the simulator retrieves the

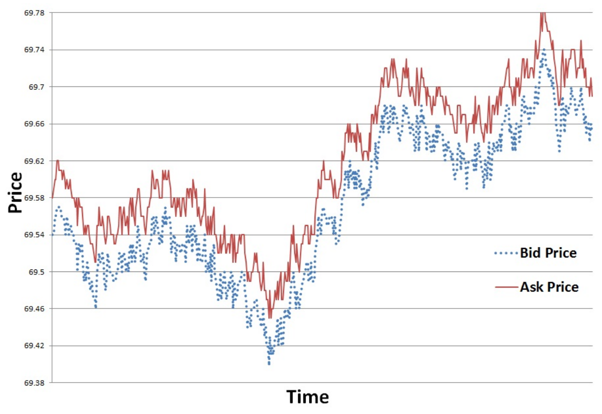


Figure 3: A simulated price movement in which spread was controlled by a Market Maker Agent. The pre-defined spread limit was set to 0.05.

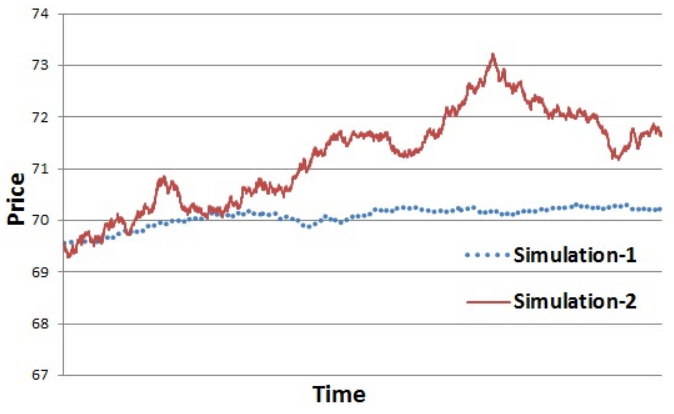


Figure 5: A comparison of two simulations with different liquidity taking.

```
8=FIX.4.2|9=201|35=D|49=Broker_16|
56=MatchingEngine|52=20140301-20:42:37.426|
34=357256|1=Sim_Account|11=8321660696624948305|
21=1|55=TD.CA|54=1|38=31600|40=2|44=93.77|
60=20140301-20:42:37.426|59=0|100=*|167=CS|10=128|
```

Figure 4: An example of FIX NewOrderSingle message. The original delimiters have been replaced by "|" for clarity. The message represents a buy limit order attempting to purchase 31,600 shares of the security *TD Canada Trust* at or below $93.77.

information by parsing the tag-value pairs and then processes the message accordingly. All returning messages, including execution reports and quotes, are also encoded as FIX format before sending back to each user. Using the FIX protocol also provides easy access to our simulator. Most users in both industrial and academic algorithmic trading settings are familiar with the FIX protocol or have it already implemented in order to connect to financial markets. This makes it possible to interact with our simulator with little modification to their systems.

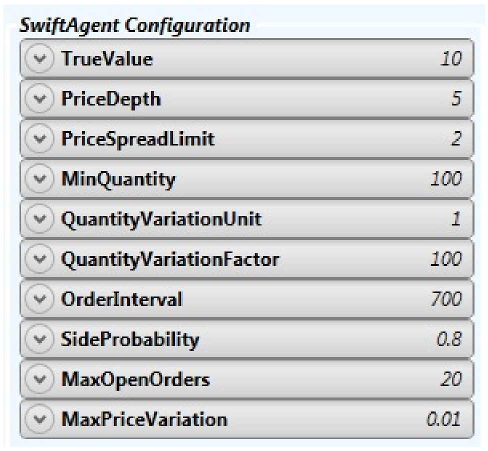
C. Simulated Trading Agents

A financial market is considered as a dynamic system constituting a heterogeneous group of investors, each following their own trading strategy in an attempt to gain superior return. A considerable amount of effort in the past years has gone into the pursuit of proper modelling of the financial market system. One well-known method, which we adopt, is to use agent-based modelling [8] where each computerized agent represents a human or robotic trader, and agents compete with each other. This determines the price of securities and consequently forms a market. Since price fluctuations can depend on the interactions of all the agents and on additional conditions that did not take place in historical data, trading strategies can be arguably more fully evaluated by this approach than in the back-testing model [9]. In order to create various market conditions that are suitable for testing, we have developed five pre-defined types of agents that represent an important subset of trading entities we observe in the real market. These agents are able to adapt to the market and interact with users' algorithmic trading strategies. We outline each of these types of agent below. Other agents may be defined by programmed extensions, but the pre-defined agents are often enough to model desired scenarios.

Market Maker Agent The Market Maker Agent plays neutrally against the market and, as its name suggests, is an imitation of the market makers that commonly exist in many exchanges, such as TSX [10] and NASDAQ [11]. A market maker's primary objective is to enhance the liquidity and the depth of the market, typically for a specific security. This provides an efficient way to get into and out of the market for small investments in the given security. Market makers also contribute to the stability of the market. When the security price is moving significantly up or down, the agent will post a reasonable volume of offers in an attempt to counter the trend. The Market Maker Agent is sensitive to price changes on both sides and keeps monitoring the difference between the bid and ask (the spread). If the difference exceeds a pre-defined threshold, it will adjust its orders accordingly to reduce the spread. Figure 3 shows a simulated price movement in which spread was monitored by a Maker Maker Agent.

Liquidity Taker Agent The Liquidity Taker Agent takes liquidity from the market by posting market orders that are often immediately executed at the best available price. This is a special type of agent as it can be used to introduce volatility to the market. By increasing the size or the frequency of the orders, it can potentially fill more orders from the opposite side, which often causes the quoted prices to change dramatically. Figure 5 shows a comparison of volatility between two simulations. The settings were the same except in the second simulation the Liquidity Taker Agent issues market orders at a higher frequency.

Liquidity Provider Agent In contrast, the Liquidity Provider Agent provides liquidity to the market by placing limit orders to the market. A buy limit order can only be executed at the specified price or lower and a sell limit order can only be executed at the specified price or higher. By posting limit orders on both sides without immediately triggering a trade, it adds liquidity to the order book and consequently increases the depth and the stability of the market.



SwiftAgent Configuration

TrueValue	10
PriceDepth	5
PriceSpreadLimit	2
MinQuantity	100
QuantityVariationUnit	1
QuantityVariationFactor	100
OrderInterval	700
SideProbability	0.8
MaxOpenOrders	20
MaxPriceVariation	0.01

Figure 6: The user interface of Swift Agent configuration.

Random Agent The Random Agent uses no information about the market and issues random orders at certain time intervals. This type of agent can be used to create chaos in the simulation environment as well as to investigate the cause of certain market phenomena. At each time a Random Agent decides whether to apply an action to the market, and if so, to which side. The order quantity varies within a specified range and its value is computed using a pseudo-random number generator. The order price follows a standard normal distribution with the mean at the current trade price and with a configurable standard deviation.

Swift Agent We have also developed a special type of agent that we call the Swift Agent. Compared to the other four agents, a Swift Agent is more sophisticated in that it is able to control the number of open orders it places. This prevents the agent from exposing itself to too much risk, just as human traders would also do in a real market. Meanwhile, by reducing the number of open orders in the matching engine, this type of agent lightens the burden on the simulation server and consequently improves overall performance. In addition, the agent is able to monitor the price fluctuations of the simulated market. If the price variation exceeds a certain threshold, the agent will attempt to place more orders on the opposite side to counter the trend.

III. IMPLEMENTATION

We have built two types of simulators based on the framework presented in Section II, and we explain their implementations below.

A. *Agent-Based Simulation*

A variety of existing agent-based simulators work in a synchronous way where all agents are placed in a waiting queue. At any given time only one agent is active and the rest are all in a "sleep" state. The active agent performs some actions against a matching engine, notifies the next agent to "wake up" and then transitions back to a "sleep" state. Thus, the next agent is not allowed to do anything until the active agent finishes its job. Clearly this typical mechanism differs significantly from real-world environments, which are continuous and asynchronous. In order to address this issue, our agent-based simulator, in contrast, supports asynchronous operation of agents so that a number of agents can perform a variety of actions at the same time. In addition, our simulator guarantees that all requests submitted by agents are processed at the precise time of their arrival. This avoids accumulating inaccuracies as the simulation evolves. Our simulator also maintains the correct ordering of the requests. This allows simulations to respect causality, and each agent can re-schedule its future actions.

Each agent provides a succinct graphical user interface for quick configuration and to adjust the parameters that define every aspect. Figure 6 shows a snapshot of the graphical user interface of the Swift Agent. By adjusting the configuration of each agent or the proportion of each agent type, one can develop a flexible, realistic, and efficient simulated market with large number of different agents. Figure 7 shows a snapshot of launching multiple agents in our simulation environment.

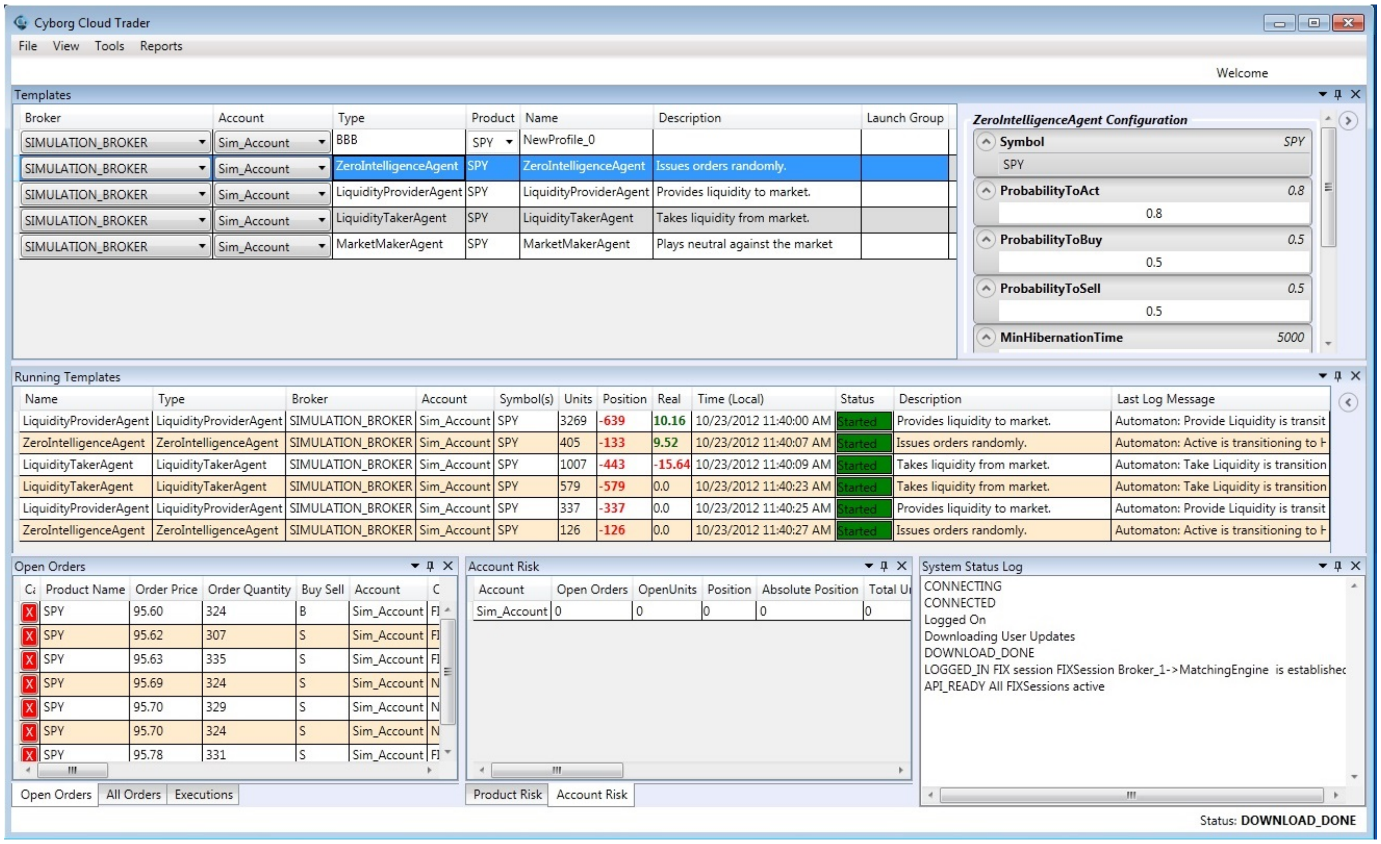
B. *Live-Data Simulation*

We now describe another type of simulator which uses live data from real exchanges. Orders received from users are matched against the current quoted prices in the market. This provides users a risk-free environment to test their trading strategies in a realistic setting (although, of course, orders in the simulation do not affect the live market). As mentioned earlier, prior real market data simulation technology already exists. However, our simulator outperforms others in several different aspects. First, it supports a variety of security types, including equities, futures, foreign exchange and options, while others typically allow only a single security type. Second, compared to existing simulators such as the Penn Exchange Simulator (see Section V), which pulls only snapshots of the current markets at certain time intervals, our simulator uses tick-by-tick data. This has a much finer granularity and therefore can reflect the current market conditions more precisely. Last, but not least, unlike existing simulators that typically require programming skills in specific languages, our simulator adopts the FIX protocol, which will ease the process of integrating the simulator into other systems.

The throughput of the matching system is extremely important. One of the biggest challenges we have encountered in the development of the simulator was the delivery of the massive volume of real-time market data to the trading agents and logged in participants. Following 100 products, we could easily hit the limit of our server, which is about 1500 quotes per second. As the simulator needs to publish the quotes to each of its subscribers, the number of quotes that actually pass through the system could be significantly larger. To improve performance, we use immutable objects in the matching engine, allowing multiple threads to use the data simultaneously without exclusive locking. We found this eliminated significant overhead. In our experiments, we found that the server was able to subscribe up to 350 products and latency was not observed after a long run.

C. *The Software*

The components of the simulator were written primarily in the C# programming language. We also adopted F# to implement the message objects that passed through the system as it has native support for immutability. In total, there were 137 classes with about 40,000 lines of code, and it took approximately 30 person-months to implement. The FIX protocol that we adopted was version 4.2, which is currently prevalent in industry. The simulation servers were deployed on Microsoft Windows 2012 servers.



Templates

Broker	Account	Type	Product	Name	Description	Launch Group
SIMULATION_BROKER	Sim_Account	BBB	SPY	NewProfile_0		
SIMULATION_BROKER	Sim_Account	ZeroIntelligenceAgent	SPY	ZeroIntelligenceAgent	Issues orders randomly.	
SIMULATION_BROKER	Sim_Account	LiquidityProviderAgent	SPY	LiquidityProviderAgent	Provides liquidity to market.	
SIMULATION_BROKER	Sim_Account	LiquidityTakerAgent	SPY	LiquidityTakerAgent	Takes liquidity from market.	
SIMULATION_BROKER	Sim_Account	MarketMakerAgent	SPY	MarketMakerAgent	Plays neutral against the market	

ZeroIntelligenceAgent Configuration

Symbol	SPY
ProbabilityToAct	0.8
ProbabilityToBuy	0.5
ProbabilityToSell	0.5
MinHibernationTime	5000

Running Templates

Name	Type	Broker	Account	Symbol(s)	Units	Position	Real	Time (Local)	Status	Description	Last Log Message
LiquidityProviderAgent	LiquidityProviderAgent	SIMULATION_BROKER	Sim_Account	SPY	3269	-639	10.16	10/23/2012 11:40:00 AM	Started	Provides liquidity to market.	Automaton: Provide Liquidity is transit
ZeroIntelligenceAgent	ZeroIntelligenceAgent	SIMULATION_BROKER	Sim_Account	SPY	405	-133	9.52	10/23/2012 11:40:07 AM	Started	Issues orders randomly.	Automaton: Active is transitioning to H
LiquidityTakerAgent	LiquidityTakerAgent	SIMULATION_BROKER	Sim_Account	SPY	1007	-443	-15.64	10/23/2012 11:40:09 AM	Started	Takes liquidity from market.	Automaton: Take Liquidity is transition
LiquidityTakerAgent	LiquidityTakerAgent	SIMULATION_BROKER	Sim_Account	SPY	579	-579	0.0	10/23/2012 11:40:23 AM	Started	Takes liquidity from market.	Automaton: Take Liquidity is transition
LiquidityProviderAgent	LiquidityProviderAgent	SIMULATION_BROKER	Sim_Account	SPY	337	-337	0.0	10/23/2012 11:40:25 AM	Started	Provides liquidity to market.	Automaton: Provide Liquidity is transit
ZeroIntelligenceAgent	ZeroIntelligenceAgent	SIMULATION_BROKER	Sim_Account	SPY	126	-126	0.0	10/23/2012 11:40:27 AM	Started	Issues orders randomly.	Automaton: Active is transitioning to H

Open Orders

Ca	Product Name	Order Price	Order Quantity	Buy Sell	Account	C
X	SPY	95.60	324	B	Sim_Account	FI
X	SPY	95.62	307	S	Sim_Account	FI
X	SPY	95.63	335	S	Sim_Account	FI
X	SPY	95.69	324	S	Sim_Account	N
X	SPY	95.70	329	S	Sim_Account	N
X	SPY	95.70	324	S	Sim_Account	N
X	SPY	95.78	331	S	Sim_Account	FI

Account Risk

Account	Open Orders	OpenUnits	Position	Absolute Position	Total U
Sim_Account	0	0	0	0	0

System Status Log

```
CONNECTING
CONNECTED
Logged On
Downloading User Updates
DOWNLOAD_DONE
LOGGED_IN FIX session FIXSession Broker_1->MatchingEngine  is establishec
API_READY All FIXSessions active
```

Status: **DOWNLOAD_DONE**

Figure 7: A snapshot of launching multiple agents.

IV. Useful Scenarios

Both the agent-based and the live-data simulator have been adopted by Quantica Trading, a company located in Kitchener, Canada, which develops algorithmic trading software. They have been deployed on production servers to provide immediate access to users both inside and external to the company. Both have been extensively used by researchers and engineers on a daily basis.

We have found, informally, the agent-based and live-data simulators to be useful in the following scenarios.

A. *Trading System and Strategy Testing*

Testing of algorithmic trading strategies can be challenging as there can be a variety of conditions that must be taken into account. Some conditions, such as a market crash, may not occur very often in a real market, but they are extraordinarily costly when they do occur and so must be examined. With the agent-based simulator, we can easily reproduce these conditions, and variants, to test strategies. Figure 8 shows a bubble created using our simulator.

We have also provided a list of special order books in our simulator, each of which applies a deterministic rule when executing orders, such as full fill, partial fill, slow fill and etc. We found they were particularly useful for testing trading systems which typically require deterministic input and output. By issuing orders on a special testing security, we can anticipate deterministic responses, which are very suitable for black-box testing.

B. *Education and Training*

Our simulator has also been found useful for education and training purposes. By executing trades in the risk-free simulation environment, it facilitates trading drills designed for new traders, allowing them to learn how to execute trades and manage risk faster.

C. *Software Demonstration*

Our simulator is also suitable for software demonstration. With round-the-clock access and risk-free testing, users can present demonstrations of their software applications at their convenience.

D. *Evaluating Regulatory Effects*

A simulator of the type we present also benefits users beyond the high frequency trading world. High quality simulation is essential to improve the regulatory environment for North American markets. At the moment, the true impact of regulation cannot be completely understood until it is in effect in the markets. This means that regulation can have unintended consequences or not achieve its desired results. High-quality simulation can help improve this, reducing risk of events such as the “Flash Crash” of 2010 [5].

V. RELATED WORK

Building a simulation environment that is suitable for evaluation of algorithmic trading strategies is a problem for which there has been considerable previous work. We highlight some notable relevant contributions here.

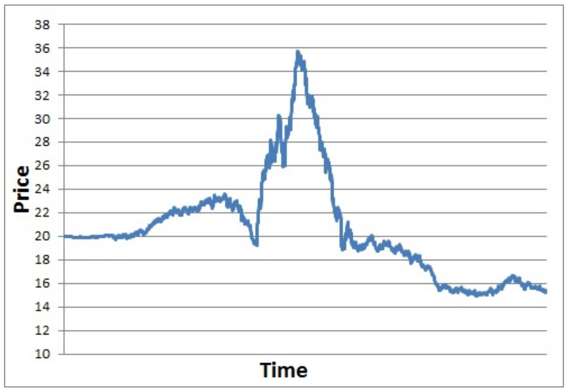


Figure 8: A bubble created by our simulator.

One of the earliest attempts was perhaps the Santa Fe Artificial Stock Market, which was initiated by Brian Arthur and John Holland. Their objective was to build a financial market with an ecological system where successful strategies would persist and replicate, and weak strategies would go away. The first version of the Santa Fe Artificial Stock Market became available in early 1990s and was published in [12]. It adopted agent-based modelling techniques but was operated naïvely. A price was announced by a market maker to all agents, then each agent submitted an order to buy or sell one share of stock. Most times, this market was out of equilibrium with either more buyers or sellers. The smaller of these two sets would get satisfied while the other would get rationed [13]. An advanced version was later developed, adding extensive modifications to the first version. It was able to generate several features similar to actual financial data. However, it did not support continuous execution mechanisms. Time was broken into discrete time periods $t = 1, 2, ...n$, in each of which agents were asked in a row to place an order in the market. This was not realistic compared to a real market where each participant can trade asynchronously. Meanwhile, human users were not allowed to interact with simulated environment.

The Penn Exchange Simulator was another software simulator for algorithmic stock trading. It received real-time order book data from Island Electronic Crossing Networks, which was provided in the form of snapshots of the top 15 limit orders (price and volume) [14]. The data was polled approximately every 3 seconds. Compared to other simulation platforms that modelled order book based on price information alone (i.e., quotes), the Penn Exchange Simulator provided a more realistic environment for simulation. For this reason, it had been chosen to be the testing platform of several Trading Agent Competitions [15]–[17]. The Penn Exchange Simulator accepted connection requests from users. In a multi-user simulation environment, the orders were matched both with other users' orders and with the orders from the real market. This allowed blending the internal and external markets. To evaluate the performance of each user, the Penn Exchange Simulator automatically computed various quantities of interest, including profit and loss. The Penn Exchange Simulator provided designated ports, to which a number of clients could connect. Each client was allowed to perform a variety of actions such as buy or sell orders, examine his or her profile, and monitor market data. On the other hand, there were several drawbacks. First, it supported limit orders only and did not allow securities other than stocks. This restricted its use to test certain trading strategies. Second, users with enough resources could potentially take advantage of the data latency by gaining access to a faster real-time data source. Last, but not least, once trading strategies were fixed, users were not allowed to intervene during the day.

The Investopedia Simulator [18] was a web-based simulator whose primary purpose was to introduce people to the stock market. It used market data from real exchanges in order to imitate the experience of dealing with a real online brokerage account. Each registered user was allocated with certain amount of virtual cash and could issue trades based on real market data. Similar to the Penn Exchange Simulator, the Investopedia Simulator was only available during market hours. In addition, as only manual orders were allowed, it was not possible to test algorithmic trading strategies.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

Recent market events, such as the "Flash Crash" and the Knight Capital trading disruption [19], have once again stressed the need to have a platform that is suitable to develop and evaluate algorithmic trading strategies. We have presented a financial market simulator that supports a full range of security types and allows users to interact as if they were trading in a real market. In addition, it adopts FIX as the communication protocol, allowing multiple users to interact with the simulation environment simultaneously and independently. We have also presented several types of simulated trading agents which represent a subset of traders observed in real markets. All of these agents are configurable and, by adjusting their parameters, very specific market conditions can be created to explore certain market behaviours. We have found that in a corporate setting that our simulator is useful in a number of scenarios, including system testing, education, training, and policy evaluation.

There are a few interesting directions that we would like to pursue in the future. First, all the securities offered by the current simulator are independent from each other, but in a real market some are correlated. That is, if the price of one security changes, the other security will move in either the same or the opposite direction. In order to create a more realistic simulation environment, we may compute how securities move in relation to each other from historical market data and use such correlations to guide the agents when they interact with the simulator. This would allow prediction of the price movement of multiple securities and consequently better evaluate overall market conditions. Second, we may wish to compare the simulated market with the actual one. This would allow us to assess not only the fidelity of the trading strategies employed by our agent models but also the possible impact if they were deployed in a real market. Last, but not least, it would be interesting to develop advising agents to provide support to human traders when trading against real markets, helping them reduce risk and optimize profit.

ACKNOWLEDGMENTS

This work was supported, in part, by a CU-I2I grant from the Natural Sciences and Engineering Research Council of Canada. We thank James McInnes, Jonathan Leaver and Travis Felker for discussions on desired functionality, and Peter Metford and Vadim Mazalov for discussions relating to implementation.

References

[1] T. Hendershott, C. M. Jones, and A. J. Menkveld, “Does Algorithmic Trading Improve Liquidity?” J. Finance, vol. 66, no. 1, Feb 2011, pp. 1–33.

[2] R. Curran and G. Rogow, “Rise of the (Market) Machines,” http://blogs.wsj.com/marketbeat/2009/06/19/rise-of-the-market-machines/, [retrieved: Sep, 2014].

[3] R. Iati, “The Real Story of Trading Software Espionage,” http://www.wallstreetandtech.com/trading-technology/the-real-story-of-trading-software-espionage/a/d-id/1262125?, [retrieved: Sep, 2014].

[4] Trading Technologies Inc., “TT SIM,” https://www.tradingtechnologies.com/ttsim/, [retrieved: Sep, 2014].

[5] U.S. Commodity Futures Trading Commission and US Securities & Exchange Commission, “Findings Regarding the Market Events of May 6, 2010,” http://www.sec.gov/news/studies/2010/marketevents-report.pdf, [retrieved: Sep, 2014].

[6] FIX Trading Community, “Financial Information eXchange (FIX) Protocol,” http://www.fixtradingcommunity.org/, [retrieved: Sep, 2014].

[7] Quantica Inc., “Simulator Demo,” http://works.bepress.com/stephen_watt/3, [retrieved: Sep, 2014].

[8] B. LeBaron, “Agent-based Computational Finance: Suggested Readings and Early Research,” J. Economics Dynamics and Control, vol. 24, no. 5-7, June 2000, pp. 679–702.

[9] N. Ponomareva and A. Calinescu, “Extending and Evaluating Agent-Based Models of Algorithmic Trading Strategies,” IEEE Int’l Conf. on Engineering of Complex Computer Systems, 2012, pp. 351–360.

[10] Toronto Stock Exchange, http://www.tmx.com/, [retrieved: Sep, 2014].

[11] NASDAQ Exchange, http://www.nasdaq.com/, [retrieved: Sep, 2014].

[12] R. G. Palmer, W. B. Arthur, J. H. Holland, B. LeBaron, and P. Tayler, “Artificial Economic Life: A Simple Model of a Stockmarket,” Physics D, vol. 75, no. 1-3, Aug. 1994, pp. 264–274.

[13] B. LeBaron, “Building the Santa Fe Artificial Stock Market,” Brandeis University, Tech. Rep., 2002.

[14] M. Kearns and L. Ortiz, “The Penn-Lehman Automated Trading Project,” IEEE Intelligent Systems, vol. 18, 2003, pp. 22–31.

[15] M. P. Wellman, A. Greenwald, P. Stone, and P. R. Wurman, “The 2001 Trading Agent Competition,” IEEE Internet Computing, vol. 13, 2000, pp. 935–941.

[16] M. P. Wellman et al., “Designing the Market Game for a Trading Agent Competition,” IEEE Internet Computing, vol. 5, no. 2, 2001, pp. 43–51.

[17] P. Stone et al., “ATTac-2000: An Adaptive Autonomous Bidding Agent,” J. Artificial Intelligence Research, vol. 15, 2001, pp. 189–206.

[18] Investopedia, “Investopedia Simulator,” http://www.investopedia.com/simulator/, [retrieved: Sep, 2014].

[19] Reuters, “Error by Knight Capital Rips Through Stock Market,” http://www.reuters.com/article/2012/08/01/us-usa-nyse-tradinghalts-idUSBRE8701BN20120801, [retrieved: Sep, 2014].